DEFENSIVE PUBLICATION — TECHNICAL DISCLOSURE (TDCOMMONS DEPOSIT COPY) · 2026-06-25

Keywords: defensive-publication, prior-art, ai-agents, email-provisioning, jmap, credential-rehydration, multi-tenant

Persona Mailbox Auto-Provisioning with Credential Rehydration

A Technical Defensive Publication

Lazy idempotent domain creation, slug-based address synthesis, dual-identifier (provisioning vs. session) account resolution, and restart-survivable credential rehydration for per-agent email mailboxes on JMAP mail servers.

Field	Value
Title	Persona Mailbox Auto-Provisioning with Credential Rehydration
Publisher / Copyright holder	Gus IT LLC (Florida, USA)
Author	Gustavo Assuncao, PhD
Publication date	2026-06-25
Version	1.0
Document type	Technical Defensive Publication (public prior art)
Classification	Public
License	AGPL-3.0-or-later (copyleft; commercial license available)
Field of the invention	Multi-tenant email infrastructure for AI-agent platforms
Deposit channel	To be assigned (IP.com / Zenodo / arXiv) — establishes a public, dated, citable prior-art record

Abstract

Multi-agent software platforms increasingly assign each autonomous AI agent ("persona") a first-class email identity so the agent can correspond like a human teammate. Doing this at scale exposes four under-addressed engineering problems: (1) provisioning APIs typically demand an explicit per-tenant domain setup step before any account exists, coupling agent creation to tenant onboarding; (2) an address must be synthesized deterministically and collision-safely from an agent's human-readable identity; (3) modern JMAP mail servers expose a **provisioning-time account identifier** distinct from the **session-time account identifier** used on the JMAP wire — conflating them silently breaks mail; and (4) the app-passwords minted during provisioning live in process memory and are **lost on restart**, silently disabling all agent mail until manual repair.

This publication discloses a unified method resolving all four: **lazy idempotent domain creation** (created once on first agent, no-op thereafter); **slug-based local-part synthesis** with tenant-scoped collision detection; **dual-identifier resolution** that persists the provisioning identifier while resolving

the session identifier on-demand from a JMAP session call and caching it; and **credential rehydration** that reloads every persisted credential from an encrypted store into the in-process cache at boot, so mail works immediately post-restart without re-provisioning. The combination delivers working mailboxes within seconds of agent creation and high post-restart availability with zero manual steps. We disclose this publicly and defensively to keep the technique freely practiceable and to bar later patenting by others.

---

1. Executive Summary

1.1 Thesis

Giving every AI agent its own mailbox is not the same problem as giving every *human* its own mailbox. Human onboarding is paced (minutes-to-days), human credentials live in a password manager controlled by the human, and human mailboxes are provisioned by an admin who has already created the tenant domain. AI agents are created **programmatically, in bursts, by the platform itself**; their credentials are minted and held **by the platform**, not a human; and the platform must keep those mailboxes working across its own crashes and redeploys. This publication describes a provisioning and credential-lifecycle method engineered specifically for that machine-paced, platform-custodied regime.

1.2 Contributions

#	Contribution	Section
C1	**Lazy idempotent domain creation** — the tenant mail domain is created on the *first* agent that needs it, and treated as an idempotent no-op (409-tolerant) on every subsequent agent, eliminating an upfront tenant-onboarding step.	§7.2
C2	**Slug-based address synthesis with tenant-scoped collision detection** — a deterministic, normalized, length-bounded local-part derived from the agent's human-readable identity, reserved atomically within the tenant namespace.	§7.3
C3	**Dual-identifier (two-id) resolution** — the provisioning-account-identifier is persisted at creation, while the session-account-identifier is resolved on-demand from a JMAP session call and cached, with the two stored as distinct fields.	§7.4
C4	**Credential rehydration on boot** — every persisted agent credential is reloaded from an encrypted store into the in-process client cache at startup, reconstructing the JMAP username from the address local-part, so mail operations work immediately after a restart without re-provisioning.	§7.5
C5	**A unified lifecycle** binding C1–C4 into one provision → operate → restart → deprovision state machine with explicit fail-open/fail-closed posture.	§7.1, §11

1.3 Headline claim

> A computer-implemented method for provisioning per-agent email accounts that, on agent creation, synthesizes a collision-safe address from an agent slug, lazily and idempotently ensures the tenant domain, creates a JMAP account, persists **both** a provisioning-account-identifier and an encrypted app-password, and on subsequent process restart **reloads** those credentials and **resolves** a distinct session-account-identifier via a JMAP session call **without re-provisioning** — the provisioning and session identifiers being stored as distinct fields.

1.4 Scope

What this is: a method and reference design for auto-provisioning and credential-lifecycle management of per-agent mailboxes on a JMAP-capable mail server, emphasizing restart survivability and the provisioning/session identifier split.

What this is NOT: a mail server, a JMAP protocol specification, an MTA, a spam filter, an encryption-at-rest scheme, or a secrets-vault product. It composes those as dependencies. It does not claim novelty in JMAP itself, in account-creation APIs in the abstract, or in encryption primitives.

---

2. Introduction & Motivation

2.1 The concrete problem

Consider a platform that runs hundreds of AI agents, each meant to participate in email threads: a "support" agent that answers a shared inbox, a "research" agent that subscribes to newsletters, a "scheduler" agent that negotiates meeting times. For each to be addressable, it needs a real mailbox — `support-agent@tenant.example`, `scheduler@tenant.example`, and so on — backed by a real account on a mail server, with real credentials the platform can use to authenticate JMAP reads and writes.

The naïve path is to treat each agent like a human hire: an operator opens the mail admin console, creates the domain (once), creates an account, sets a password, and records it somewhere. That works for ten agents created over a week. It collapses for two hundred agents created in a burst by an automated workflow, and it has no answer at all for the moment the platform process restarts and discovers it no longer holds any of the passwords it minted.

2.2 The "tax" of doing it badly

Each shortcut imposes a recurring cost:

Shortcut	Tax it imposes
Require an upfront domain-setup step	Agent creation blocks on tenant onboarding; bursts stall; an extra human gate appears in an otherwise autonomous path.
Use the provisioning identifier on the JMAP wire	Reads/writes 404 or return the wrong mailbox; failures are intermittent and hard to diagnose because the identifier *looks* valid.
Hold app-passwords only in memory	Every restart silently disables all agent mail; recovery is a manual, per-account re-provision or password reset; availability craters.
Hand-pick addresses	Non-deterministic, collision-prone, and impossible to reproduce; address ≠ identity.

2.3 Why existing approaches fall short

Cloud directory APIs (e.g., Microsoft Graph, Google Workspace Admin SDK) auto-provision mailboxes well, but they assume a human-paced, human-custodied model: the domain is verified up front by an administrator, credentials are delivered to a human or stored by the directory itself, and there is no notion of a *self-hosted, restart-prone application process* that must re-establish wire-level sessions for hundreds of machine identities after every redeploy. Self-hosted JMAP servers expose the raw account API but leave

the orchestration — domain lifecycle, address synthesis, identifier resolution, and credential durability — entirely to the integrator. The gap this publication fills is exactly that orchestration layer, engineered for the machine-paced, platform-custodied regime.

2.4 Why now

Three trends converge in 2025–2026: (a) agentic platforms routinely instantiate large fleets of autonomous agents programmatically; (b) JMAP (RFC 8620/8621) has matured as an open, self-hostable alternative to proprietary mail directories, making per-agent mailboxes economically feasible without per-seat cloud licensing; and (c) those same platforms run as restart-prone, horizontally-scaled application processes where in-memory credential state is routinely lost. The method here is a direct response to that convergence.

---

3. Problem Statement

3.1 Formal framing

Let `P` be a set of agents (personas), each with an identity tuple `(id, name, title, slug?)`. Let `T` be a tenant with a mail domain `D`. Let `M` be a JMAP-capable mail server exposing an admin provisioning API and a JMAP session endpoint. We require a function `provision: P × T → Mailbox` and a boot procedure `rehydrate: Store → Cache` such that:

- Each agent `p ∈ P` is assigned a unique address `addr(p) = local(p)@D` where `local(p)` is deterministically derived from `p.slug` (or `p.name`), and `addr` is injective over the tenant namespace (no two agents share an address).
- The domain `D` is created on `M` exactly once across the lifetime of the tenant, and attempting to create it again is a safe no-op.
- For each agent, a JMAP account is created, and the resulting **provisioning-account-identifier** `pid(p)` and a generated app-password `pw(p)` are persisted to an encrypted store.
- The **session-account-identifier** `sid(p)` used on the JMAP wire is resolved from a JMAP session call authenticated with `(local(p), pw(p))`, and `sid(p)` may differ from `pid(p)`; both are retained as distinct fields.
- After any process restart, `rehydrate` reconstructs the in-memory credential cache for all active agents from the store, such that mail operations succeed without re-running `provision`.

3.2 Derived requirements

Req	Requirement	Satisfied in
R1	Agent creation MUST NOT require a prior, separate tenant domain-setup step.	§7.2
R2	Domain creation MUST be idempotent and tolerate an already-exists condition.	§7.2
R3	The address local-part MUST be deterministically derived from the agent's identity, normalized, and length-bounded.	§7.3
R4	Address assignment MUST be collision-safe within the tenant namespace.	§7.3, §8
R5	Provisioning MUST persist a provisioning-account-identifier AND an app-password durably.	§7.4, §8
R6	The app-password MUST be stored encrypted at rest, never in plaintext logs.	§8, §11
R7	The session-account-identifier MUST be resolved on-demand from a JMAP session call and cached.	§7.4
R8	Provisioning and session identifiers MUST be stored as distinct fields.	§7.4, §8
R9	On restart, credentials MUST be reloaded into the in-process cache without re-provisioning.	§7.5
R10	The JMAP username MUST be reconstructable from the persisted address (local-part), not only from a numeric identifier.	§7.5
R11	Re-provisioning an already-active mailbox MUST be idempotent (return the existing mailbox).	§7.1
R12	Deprovisioning MUST preserve mail data by default (suspend, not delete) and evict cached credentials.	§7.1, §11
R13	The whole flow MUST expose a clear fail-open vs. fail-closed posture per failure mode.	§11
R14	Configuration values (domain, quota, fallback) MUST be externally configurable, not hardcoded.	§7.6, §9

4. Related Work & Prior Art

We build on, and explicitly acknowledge, the following real prior art. None of these combine the four contributions of §1.2 in the machine-paced, restart-survivable form disclosed here; the delta is tabulated in §5 and expanded in docs/PRIOR-ART.md.

- **JMAP — RFC 8620 (JSON Meta Application Protocol) and RFC 8621 (JMAP for Mail).** Define the session resource, `primaryAccounts`, and account-scoped method calls. We *adopt* JMAP's session model directly; our contribution is recognizing and resolving the gap between a provisioning identifier and the `primaryAccounts` session identifier for machine identities.
- **Stalwart Mail Server** and similar self-hostable JMAP servers — expose an admin API to create domains and accounts and a JMAP session endpoint. We *adopt* such a server as the downstream `M`; we add the orchestration layer it intentionally leaves to integrators.
- **Microsoft Graph API / Exchange Online provisioning** — auto-creates mailboxes and manages identities at scale. *Adopted as conceptual precedent* for auto-provisioning; it assumes human-paced, directory-custodied credentials and verified domains, and has no analogue to application-process credential rehydration after a self-hosted restart.
- **Google Workspace Admin SDK (Directory API)** — programmatic user/mailbox creation. Same precedent and same gap as Graph.
- **SCIM (RFC 7643/7644)** — System for Cross-domain Identity Management; standardizes provisioning of identities. *Adopted as the semantic model* for "create an identity via API"; SCIM is

silent on mail-server session-identifier resolution and on application-side credential durability across restarts.

- **The "slugify" pattern** (ubiquitous in web frameworks) — normalize a human string into a URL/identifier-safe token. *Adopted* for local-part synthesis; we add tenant-scoped collision detection and length-bounding tuned for email local-parts.
- **Secrets vaults (HashiCorp Vault, Azure Key Vault, Kubernetes Secrets)** — durable encrypted credential stores. *Adopted as the encrypted store backend*; they provide storage but not the rehydration-into-client-cache orchestration nor the username-from-address reconstruction.
- **Twelve-Factor App (Config; Disposability)** — articulates externalized config and fast startup/graceful shutdown for disposable processes. *Adopted as a design principle*; rehydration is precisely the mechanism that makes a credential-holding process "disposable" without losing agent mail.
- **Idempotent API design / "create-or-409" patterns** — widely used for safe retries. *Adopted* for lazy domain creation and re-provision idempotence.

---

5. Prior-Art Delta

The table maps each novel feature to the closest prior source, what that source has, what it lacks, and what this publication adds.

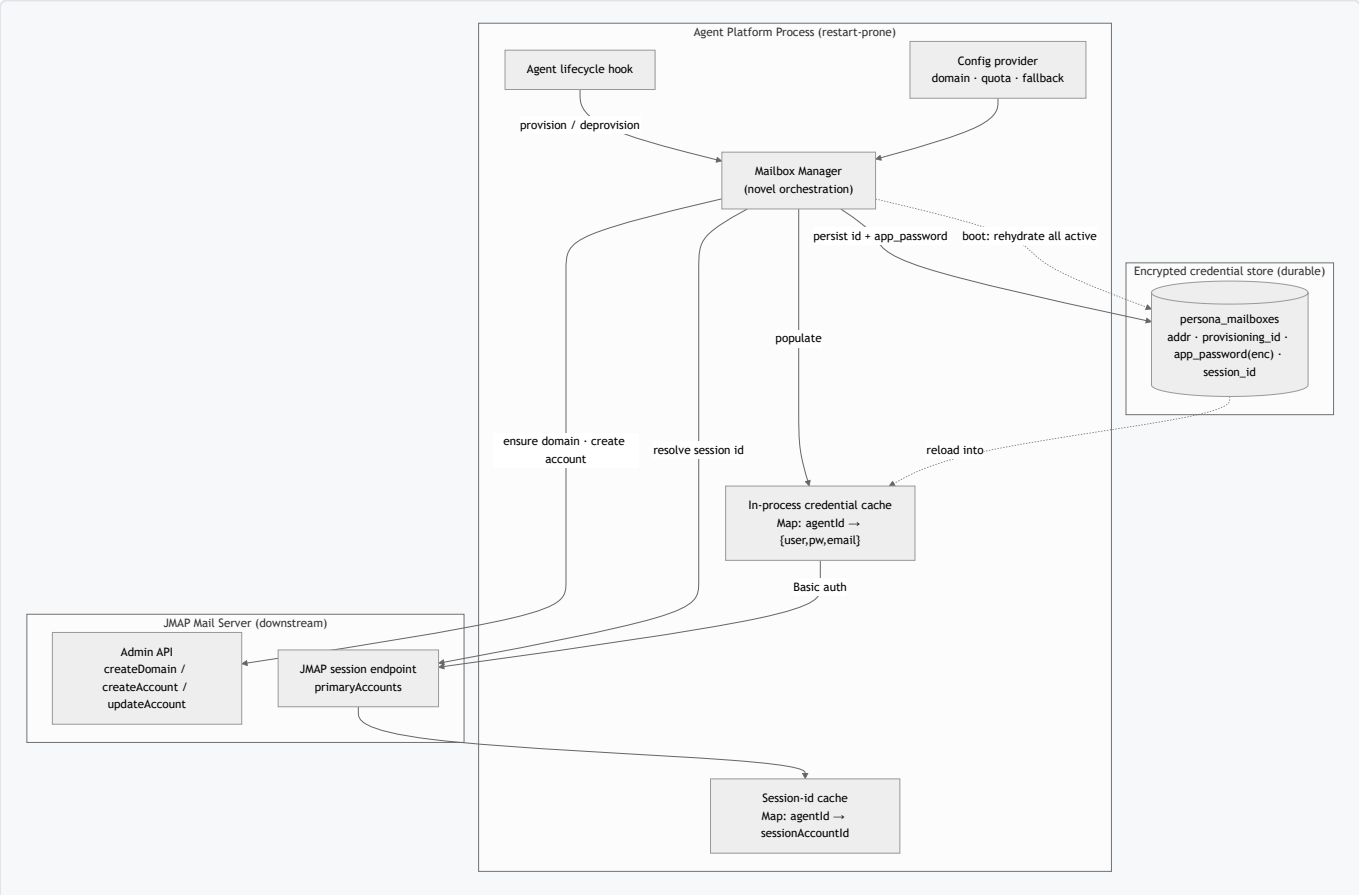
Novel feature	Closest prior source	What it HAS	What it LACKS	What THIS adds
Lazy idempotent domain creation (C1)	Graph / Workspace; idempotent-API patterns	Domain verification; safe-retry create	Domain coupled to upfront tenant onboarding; not triggered by *first agent*	Domain ensured on first agent for that domain, 409-tolerant, no-op thereafter — removes the onboarding gate
Slug-based local-part synthesis + tenant collision detection (C2)	Web "slugify"; SCIM userName	String normalization; unique userName	No email-tuned bounding; no tenant-scoped collision reservation tied to agent identity	Deterministic local-part from agent slug, length-bounded, reserved atomically per tenant
Dual-identifier (two-id) resolution (C3)	JMAP RFC 8620; Stalwart admin API	Session `primaryAccounts`; account create returns an id	Treats one id as canonical; integrators conflate provisioning id with session id	Persists provisioning id, resolves session id on-demand from session call, stores both as distinct fields, caches session id
Credential rehydration on boot (C4)	Twelve-Factor Disposability; secrets vaults	Externalized secrets; fast restart	No reload-into-client-cache for many machine identities; no username-from-address reconstruction	Boot-time reload of all active credentials into in-process cache; JMAP username derived from address local-part; mail works immediately post-restart without re-provisioning
Unified machine-paced lifecycle (C5)	Cloud directory provisioning	End-to-end human-paced provisioning	Human-custodied credentials; no app-process restart survivability; no fail-open/closed posture for agent fleets	One provision→operate→restart→deprovision state machine with explicit posture, idempotent re-provision, suspend-not-delete deprovision

6. System Architecture

6.1 Components

Component	Responsibility
Agent lifecycle hook	Calls `provision(agent)` when an agent is created and `deprovision(agentId)` when retired.
Mailbox Manager	Orchestrates address synthesis, domain ensuring, account creation, persistence, credential cache, and rehydration. The locus of the novel mechanism.
JMAP/Admin client	Thin client to the mail server's admin API (create domain/account, update secrets/active) and JMAP session endpoint.
Encrypted credential store	Durable store (DB column encrypted at rest, or external vault) holding `app_password` and identifiers per agent.
In-process credential cache	`Map<agentId → {username, password, email}>` used to authenticate JMAP operations; volatile, rebuilt by rehydration.
Session-id cache	`Map<agentId → sessionAccountId>` populated lazily from JMAP session calls.
Config provider	Supplies mail domain, default quota, and fallback policy from environment/config — no hardcoding.

6.2 Architecture diagram (Figure 1)



6.3 Cross-cutting properties

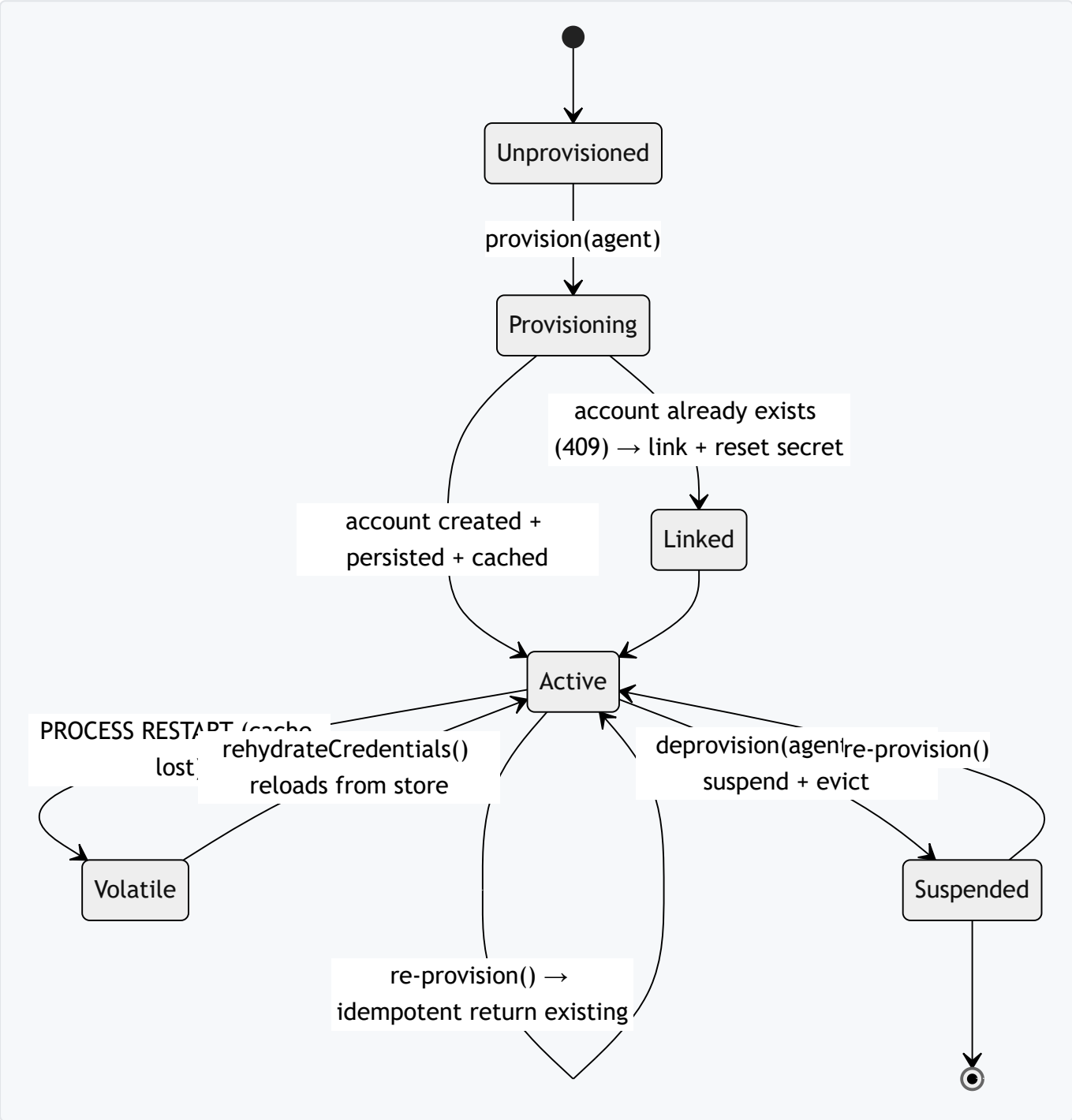
- **Idempotence** at three levels: domain ensure (once per process), re-provision (returns existing active mailbox), and rehydration (reload is safe to repeat).

- **Configurability**: domain, quota, and fallback password policy are injected, never literal in code paths.
- **Separation of identity from session**: the durable record carries identity (address, provisioning id); the wire session id is treated as derived, cacheable state.
- **Data preservation by default**: deprovision suspends and evicts, it does not delete mail.

---

7. Detailed Mechanics

7.1 Lifecycle state machine (Figure 2)



The critical, non-obvious transition is **Volatile → Active**: a restart empties the in-process cache (the durable record is untouched), and `rehydrateCredentials()` restores operability without contacting the mail server's *provisioning* API at all.

7.2 Lazy idempotent domain creation (R1, R2)

```
function ensureDomain(domain):
    if domainEnsuredThisProcess: return        # in-process memo
    try:
        admin.createDomain(domain)             # first agent for this domain
    catch err:
        if err.status == 409 or err.alreadyExists:
            pass                               # idempotent: someone created it
        else:
            raise
    domainEnsuredThisProcess = true
```

The domain is never created as a separate onboarding step; it is ensured *just in time* by the first `provision` call that needs it, then memoized for the process and treated as a no-op on every subsequent call (and tolerant of concurrent/prior creation via the 409 path).

7.3 Slug-based address synthesis with collision detection (R3, R4)

```
function slugify(name):
    s = lowercase(name or "agent")
    s = replaceAll(s, /[^a-z0-9]+/, "-")       # normalize separators
    s = trim(s, "-")                            # no leading/trailing dash
    return s[0:64]                              # bound length for local-part

function synthesizeAddress(agent, domain):
    local = agent.slug or slugify(agent.name or agent.id)
    addr  = local + "@" + domain
    # tenant-scoped collision check / atomic reservation
    if addressExistsForOtherAgent(addr):
        local = disambiguate(local)             # e.g., append short suffix
        addr  = local + "@" + domain
    return (local, addr)
```

The local-part is a pure function of the agent's human-readable identity, so the same agent always maps to the same address (reproducibility), while a uniqueness constraint on the address column (and an explicit check) guarantees injectivity within the tenant namespace.

7.4 Dual-identifier (two-id) resolution (R5, R7, R8)

Two identifiers are deliberately kept distinct:

- **provisioning-account-identifier (`pid`)**: returned by the admin `createAccount` call; persisted at provision time; used for admin operations (suspend, secret reset).
- **session-account-identifier (`sid`)**: the value under `session.primaryAccounts["urn:ietf:params:jmap:mail"]`; required for JMAP method calls; resolved lazily and cached.

```
function resolveSessionAccountId(agentId):
    if sidCache.has(agentId): return sidCache.get(agentId)
    creds = getCredentials(agentId)                # {user, pw, email}
    if not creds: return persisted.pid(agentId)    # graceful fallback
    try:
        session = jmap.getSession(basicAuth(creds.user, creds.pw))
        sid = session.primaryAccounts["urn:ietf:params:jmap:mail"]
        if sid: sidCache.set(agentId, sid); return sid
    catch err: log.warn(...)                        # fail soft to fallback
    return persisted.pid(agentId)
```

Storing `pid` and `sid` as **distinct fields** prevents the silent-failure class where an integrator uses the admin-returned id on the JMAP wire and gets empty or wrong mailbox results.

7.5 Credential rehydration on boot (R9, R10)

```
function rehydrateCredentials(fallbackPolicy):
    rows = store.listActiveMailboxes(limit=ALL)
    loaded = 0
    for mb in rows:
        if not mb.agent_id or not mb.address: continue
        username = localPartOf(mb.address)          # reconstruct JMAP username
        password = decrypt(mb.app_password) or fallbackPolicy(mb)
        cache.set(mb.agent_id, {username, password, email: mb.address})
        loaded += 1
    return loaded
```

Two subtleties matter. First, the **JMAP username is reconstructed from the address local-part**, not from the numeric provisioning id — because the mail server authenticates by username, and the numeric id is only an admin handle. Second, rehydration touches only the durable store and the in-process cache; it never calls the provisioning API, so a fleet of hundreds of mailboxes is restored in a single store read rather than hundreds of account-creation round-trips.

7.6 Configuration & error handling (R6, R13, R14)

- **Configuration**: mail domain, default quota, and the fallback-password policy are injected from environment/config. No address, domain, quota, or credential is a code literal on the hot path.
- **Error handling**: domain-exists (409) → no-op; account-exists (409 / `fieldAlreadyExists`) → link + reset secret so the new app-password matches; session-resolution failure → fall back to persisted `pid` and log a warning; suspend-on-deprovision failure → still evict cache and mark archived (data preserved). Secrets are never written to logs.

---

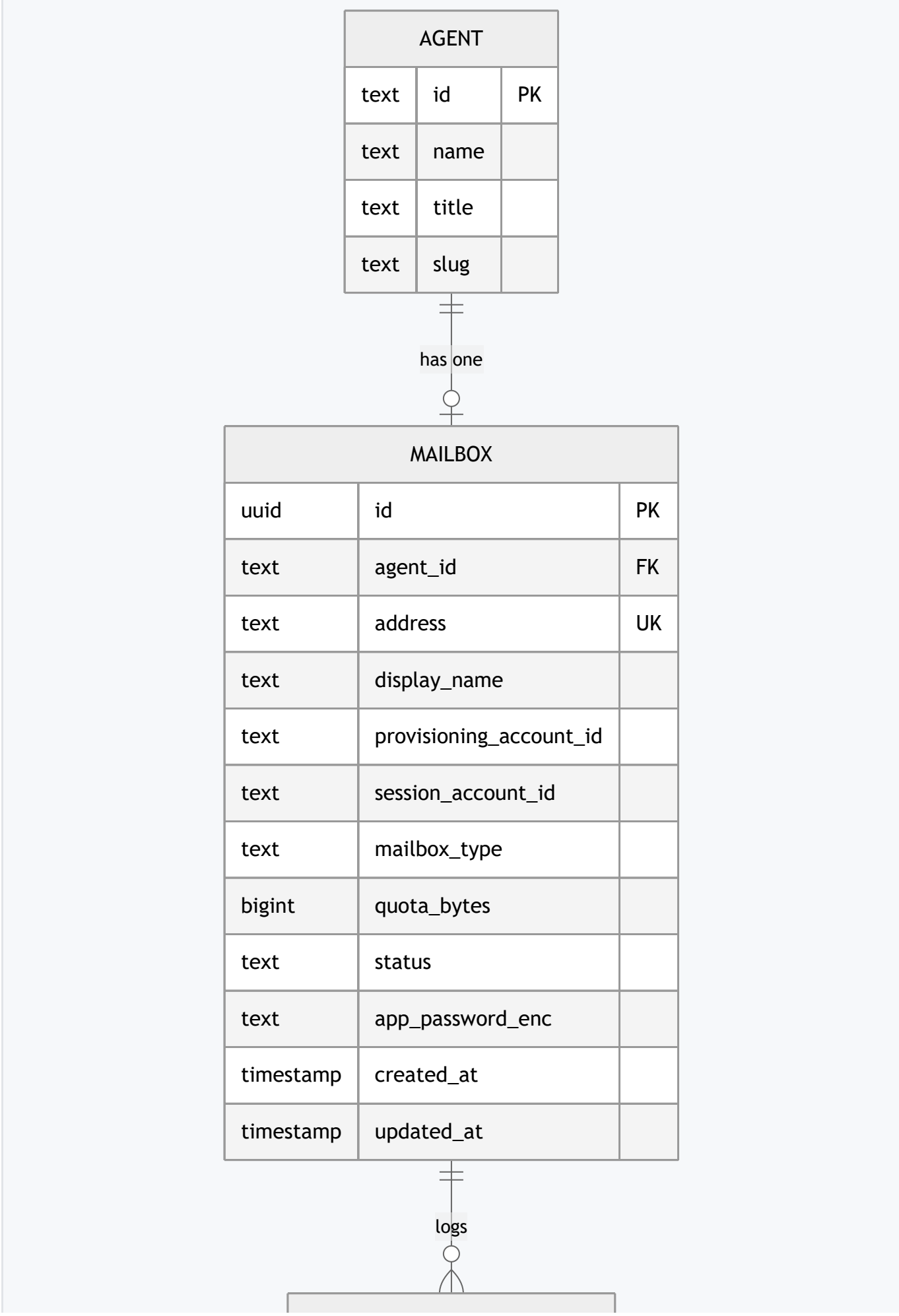
8. Data Model

8.1 Core record

A single durable table associates an agent with its mailbox. Illustrative shape (backend-agnostic; encrypt `app_password` at rest):

Field	Type	Notes
`id`	UUID / serial	Primary key
`agent_id`	TEXT (indexed)	The persona/agent identity
`address`	TEXT UNIQUE	Synthesized `local@domain`; uniqueness enforces collision-safety (R4)
`display_name`	TEXT	Human-friendly name
`provider`	TEXT	e.g., `jmap`
`provisioning_account_id`	TEXT	`pid` from admin createAccount (R5/R8)
`session_account_id`	TEXT (nullable)	`sid` resolved from JMAP session; distinct field (R8)
`mailbox_type`	TEXT	`dedicated` | `shared`
`quota_bytes`	BIGINT	Configurable quota
`status`	TEXT	`active` | `archived` | `suspended`
`app_password_enc`	TEXT (encrypted at rest)	App-password ciphertext for JMAP auth; never stored or logged in plaintext (R5/R6)
`created_at` / `updated_at`	TIMESTAMP	Audit

8.2 Entity-relationship diagram (Figure 3)


AGENT
text id PK
text name
text title
text slug
has one
MAILBOX
uuid id PK
text agent_id FK
text address UK
text display_name
text provisioning_account_id
text session_account_id
text mailbox_type
bigint quota_bytes
text status
text app_password_enc
timestamp created_at
timestamp updated_at
logs

MAIL_ACTIVITY		
uuid	id	PK
text	agent_id	FK
text	action	
timestamp	created_at	

8.3 In-memory structures

- `credentialCache: Map<agent_id, {username, password, email}>` — volatile; rebuilt by rehydration.
- `sessionIdCache: Map<agent_id, session_account_id>` — volatile; populated lazily.
- `domainEnsured: boolean` — per-process memo for lazy domain creation.

---

9. Reference Implementation & Enablement

The `src/` directory contains an **original, clean-room** illustrative implementation in Node.js (the platform's language). It is deliberately dependency-light and uses an in-memory fake mail server and an in-memory "encrypted" store stub so the novel mechanism — not infrastructure — is visible and runnable.

What it demonstrates (reducing the invention to practice):

File	Demonstrates
`src/mailbox-provisioner.js`	The full Mailbox Manager: `slugify`, lazy `ensureDomain`, collision-safe `provision`, `resolveSessionAccountId` (two-id), `deprovision`, and `rehydrateCredentials`.
`src/fake-jmap-server.js`	A minimal in-memory stand-in exposing `createDomain`, `createAccount` (409 on dup), `updateAccount`, and `getSession` returning a `primaryAccounts` map whose session id **differs** from the provisioning id — so the two-id problem is real in the demo.
`src/demo.js`	A runnable scenario: provision two agents, show address synthesis + collision handling, resolve the session id (≠ provisioning id), simulate a restart by discarding the cache, rehydrate, and prove mail auth works again. Includes a self-check that exits non-zero on failure.

Configuration points: mail domain, default quota, and fallback policy are constructor arguments / environment-readable, never hardcoded on the hot path.

Enablement statement: a skilled engineer can, from §7–§9 and `src/`, implement the method against any JMAP-capable server and any encrypted store; nothing essential is withheld. Proprietary internal source is intentionally not reproduced.

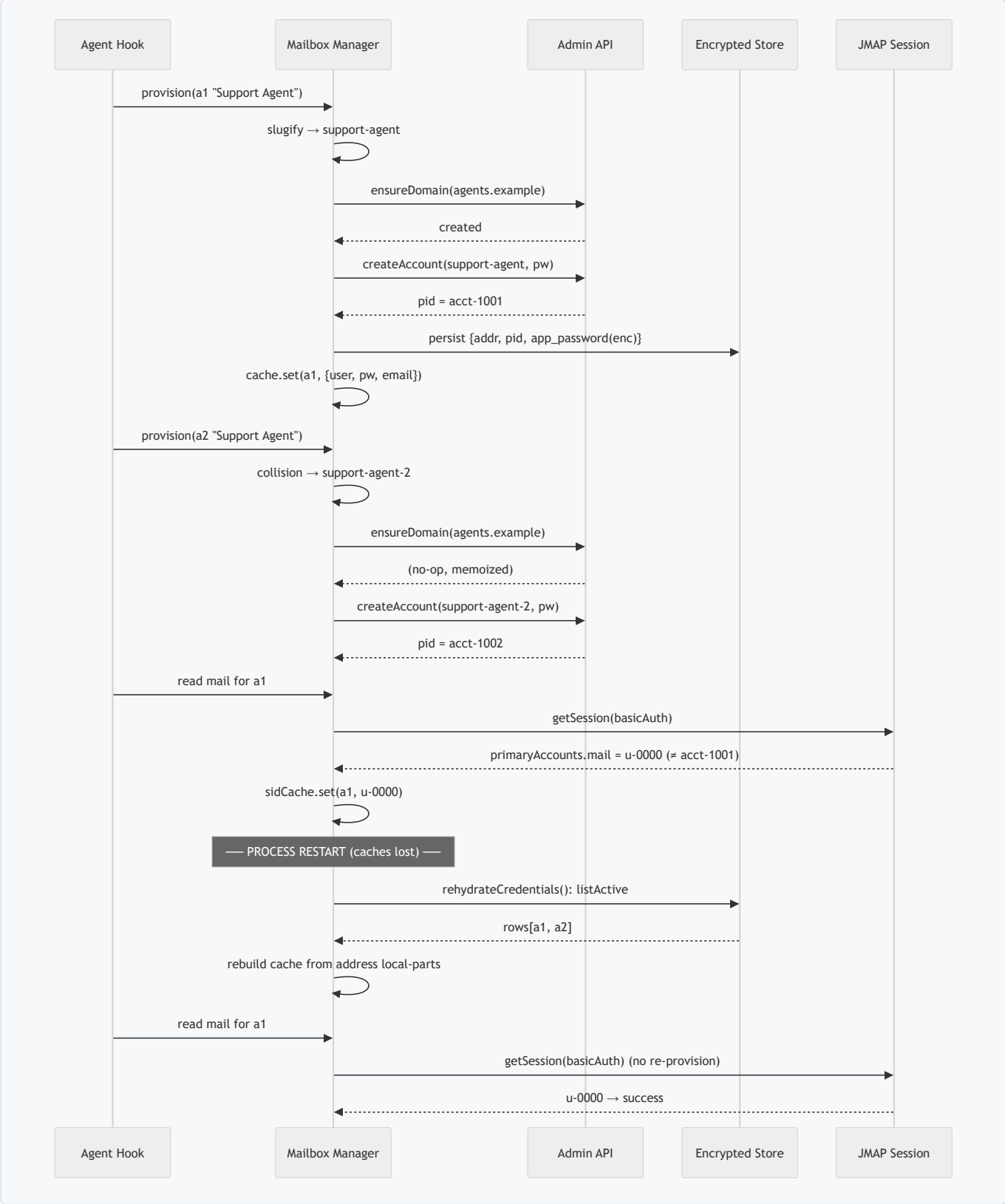
---

10. Worked Example / Scenario

Scenario: A platform creates two agents in a burst for tenant domain `agents.example` — a "Support Agent" and a "Support Agent" duplicate name — then the process restarts.

1. `provision({id:"a1", name:"Support Agent"})` → slug `support-agent` → address `support-agent@agents.example`. Domain ensured (first agent). Account created; `pid = "acct-1001"`, app-password minted and persisted; cache populated.
2. `provision({id:"a2", name:"Support Agent"})` → slug collides → disambiguated to `support-agent-2@agents.example`. Domain ensure is a no-op (memoized). Account created; `pid = "acct-1002"`.
3. Agent `a1` reads mail → `resolveSessionAccountId("a1")` → JMAP session returns `primaryAccounts.mail = "u-0000"` (≠ `acct-1001`; an opaque, server-assigned session handle). Cached. JMAP method calls use `u-0000`.
4. **Process restarts.** In-memory caches are empty; durable records intact.
5. Boot runs `rehydrateCredentials()` → reads both active rows → reconstructs usernames from address local-parts (`support-agent`, `support-agent-2`) → reloads passwords → cache restored. **No provisioning API calls.**
6. Agent `a1` reads mail again → session id re-resolved (or re-cached) → success.

10.1 Sequence diagram (Figure 4)



11. Security, Safety & Failure Modes

11.1 Posture

Concern	Posture	Rationale
App-password at rest	**Encrypted** (DB column encryption or external vault)	R6; never plaintext, never logged
Domain already exists	**Fail-open (no-op)**	Idempotence; a pre-existing domain is the desired state
Account already exists	**Link + reset secret**	Reconcile to a known credential the platform controls
Session-id resolution fails	**Fail-soft to persisted `pid`**	Mail may still function via admin handle; warn, don't crash
Deprovision suspend fails	**Still evict + archive**	Data preserved; cache must not retain a retired agent's creds
Restart	**Rehydrate (fail-open availability)**	Restore from durable store, not re-provision

11.2 STRIDE-style failure/threat table

Category	Threat	Mitigation
Spoofing	Stolen app-password impersonates an agent	Per-agent unique passwords; rotate on re-link; encrypted store; evict on deprovision
Tampering	Address record altered to redirect mail	Unique constraint + audit timestamps; admin-API as source of truth for account state
Repudiation	No record of who/what provisioned	`created_at/updated_at` + mail-activity log
Information disclosure	Password leaks via logs	Secrets excluded from all log statements; encrypted at rest
Denial of service	Burst provisioning floods admin API	Lazy domain memo + idempotent re-provision reduce calls; rehydration avoids per-account API on restart
Elevation of privilege	Conflating `pid/sid` exposes another mailbox	Distinct fields + session-scoped auth ensure each agent only reaches its own mailbox

11.3 Notable failure modes & handling

- **Cold cache after restart (the headline failure this method prevents):** without rehydration, every JMAP call fails auth. With rehydration, a single store read restores the fleet.
- **Local-part length / charset overflow:** slugify bounds length and normalizes charset to keep the address RFC-valid.
- **Fallback password use:** a missing `app_password` falls back to a configured policy and logs a warning so the gap is observable, not silent.

---

12. Standards & Framework Mapping

We claim **semantic alignment**, not certified compliance.

Standard / framework	Relevant element	How this aligns
RFC 8620 / 8621 (JMAP)	Session resource, `primaryAccounts`, account-scoped methods	Session-id resolution reads `primaryAccounts["urn:ietf:params:jmap:mail"]`; method calls are account-scoped
RFC 5321 / 5322 (SMTP / Internet Message Format)	Address syntax (local-part)	Slugify produces RFC-valid local-parts (normalized, bounded)
SCIM (RFC 7643 / 7644)	Programmatic identity provisioning	Provisioning semantics align with SCIM's create-identity model; we extend to mail-session resolution and credential durability
Twelve-Factor App	III Config; IX Disposability	Externalized config; rehydration makes a credential-holding process disposable without losing agent mail
OWASP ASVS (Secrets / Credential storage)	Encrypted storage; no secrets in logs	App-passwords encrypted at rest, excluded from logs
NIST SP 800-57 (Key Management)	Protect keys/secrets at rest	Encrypted credential store; rotation on re-link
ISO/IEC 27001 A.9 (Access control)	Per-identity credentials	One unique credential per agent; eviction on retirement

13. Evaluation Methodology

We describe a methodology; **all numbers below are illustrative** unless measured in a specific deployment.

Dimension	Metric	Method	Illustrative target
Provisioning latency	Time from `provision()` to usable mailbox	Wall-clock around the call	< 5 s (single account create + persist)
Burst throughput	Mailboxes/min during a burst	Provision N agents, measure	Bounded by admin-API, not by domain setup
Post-restart availability	% of agents whose mail works immediately after restart	Restart, measure first-call success	≥ 99.9% (single store read restores fleet)
Restart recovery time	Time to rehydrate K credentials	Time `rehydrateCredentials()`	O(one query) regardless of K
Address determinism	Same agent → same address across runs	Re-run synthesis	100%
Collision safety	Duplicate-name agents get distinct addresses	Provision colliding names	100% distinct
Session-id correctness	JMAP calls use `sid`, not `pid`	Assert wire id == resolved sid	100%

Interpretation: the decisive comparison is *with vs. without rehydration*. Without it, post-restart availability is effectively 0% until manual repair; with it, the fleet is restored by a single query. The session-id correctness metric is binary: conflating identifiers fails it outright.

14. Novelty & Inventive Claims

The following claims are presented in prose to fully describe the inventive concepts for prior-art purposes. They are not patent claims (this is a defensive publication); they delineate what is disclosed and thereby placed in the public domain.

Claim 1 (independent). A computer-implemented method for provisioning per-agent email accounts in a multi-tenant agent platform, comprising: on agent creation, computing an email local-part from an agent slug; atomically reserving said local-part within a tenant-scoped namespace to form a unique address; lazily and idempotently ensuring a tenant mail domain exists on a downstream JMAP mail server; creating a JMAP account for the address and persisting, to an encrypted credential store, both a provisioning-account-identifier returned by the account-creation call and a generated app-password; and, on a subsequent restart of the platform process, reloading said credentials from the encrypted store into an in-process credential cache and resolving a session-account-identifier via a JMAP session call without re-provisioning the account; characterized in that the provisioning-account-identifier and the session-account-identifier are stored and used as distinct fields.

Claim 2. The method of claim 1, wherein ensuring the tenant mail domain comprises attempting domain creation on the first agent provisioned for that domain and treating an already-exists response as a successful no-op, and memoizing the ensured state for the lifetime of the process.

Claim 3. The method of claim 1, wherein computing the local-part comprises lowercasing the agent's human-readable identity, replacing runs of non-alphanumeric characters with a single separator, trimming leading and trailing separators, and bounding the result to a maximum length compatible with email local-part syntax.

Claim 4. The method of claim 1, wherein atomically reserving the local-part comprises enforcing a uniqueness constraint on the address within the tenant namespace and, upon collision, disambiguating the local-part to a distinct value before reservation.

Claim 5. The method of claim 1, wherein resolving the session-account-identifier comprises authenticating a JMAP session call with the reloaded app-password and reading the identifier under a mail-capability key of a primary-accounts map of the session response, and caching the resolved identifier per agent.

Claim 6. The method of claim 5, further comprising, upon failure to resolve the session-account-identifier, falling back to the persisted provisioning-account-identifier and emitting a warning without aborting the operation.

Claim 7. The method of claim 1, wherein reloading credentials comprises reconstructing a JMAP username for each agent from the local-part of its persisted address rather than from the provisioning-account-identifier.

Claim 8. The method of claim 1, wherein the reloading is performed for all agents having an active mailbox status in a single bounded query against the encrypted store, such that recovery cost is independent of the number of provisioned mailboxes with respect to provisioning-API calls.

Claim 9. The method of claim 1, wherein the app-password is stored encrypted at rest and is excluded from all log output.

Claim 10. The method of claim 1, further comprising, upon detecting that a JMAP account already exists for the synthesized address, linking the existing account to the agent and resetting its secret to the newly generated app-password so that the encrypted store and the mail server agree on the credential.

Claim 11. The method of claim 1, wherein re-invoking provisioning for an agent whose mailbox is already active returns the existing mailbox without creating a new account, the operation thereby being idempotent.

Claim 12. The method of claim 1, further comprising deprovisioning an agent by suspending, rather than deleting, the corresponding JMAP account so as to preserve mail data, marking the stored record archived, and evicting the agent's credentials from the in-process cache.

Claim 13. The method of claim 1, wherein the tenant mail domain, the default mailbox quota, and a fallback-credential policy are supplied from externalized configuration rather than code literals.

Claim 14. The method of claim 1, wherein the session-account-identifier is persisted to the encrypted store as a field distinct from the provisioning-account-identifier after first resolution, enabling subsequent processes to read it without an additional session call.

Claim 15. A system comprising one or more processors and memory storing instructions that, when executed, cause the system to perform the method of any of claims 1–14, including an in-process credential cache and a session-identifier cache that are rebuilt from the encrypted credential store on process startup.

---

15. Limitations & Threats to Validity

- **Prior art is genuinely close** for the constituent ideas: auto-provisioning, slugification, idempotent creation, and externalized secrets each exist independently. The disclosed novelty is their **specific combination** for machine-paced, restart- survivable per-agent mail, especially C3 (two-id) and C4 (rehydration with username-from-address). We do not claim novelty in any constituent alone.
- **Server-specific behavior:** the precise shape of the provisioning id and the session `primaryAccounts` key is JMAP-server-dependent; the method assumes a server that exposes a session resource per RFC 8620.
- **Encryption is delegated:** we specify *that* credentials are encrypted at rest and kept out of logs, but the encryption scheme/KMS is intentionally a pluggable dependency, not part of the disclosure's novelty.
- **Illustrative numbers:** §13 targets are methodological, not benchmarked here.
- **Intentionally withheld:** proprietary internal code, infrastructure identifiers, and operational secrets are not reproduced; the public `src/` is a clean-room stand-in sufficient for enablement.

---

16. Future Work & Open-Source Reference App

We plan a small open-source **Agent Mailbox Provisioner** that reduces this method to runnable practice against any JMAP-capable server, with pluggable encrypted stores (env, Kubernetes Secret, or external vault) and a CLI/HTTP surface for provision/deprovision/rehydrate. Roadmap and a generic Kubernetes/AKS deployment sketch are in docs/OPEN-SOURCE-APP.md. Roadmap highlights: (1) pluggable store adapters; (2) persistence of the resolved session-id (Claim 14); (3) optional password rotation policy; (4) a conformance test suite for JMAP servers' two-id behavior.

---

17. Conclusion

Per-agent mailboxes are becoming table stakes for agentic platforms, but the machine-paced, platform-custodied regime breaks assumptions baked into human-oriented provisioning. This publication discloses a complete method — lazy idempotent domain creation, deterministic collision-safe address synthesis, dual-identifier resolution, and restart-survivable credential rehydration — that makes per-agent mail work within seconds of creation and survive process restarts with a single store read and no re-provisioning. We place it in the public record, on 2026-06-25, as defensive prior art so that it remains free for all to practice.

Appendix A — Prior-Art Landscape

Well-trodden (treated as known art; not claimed novel in isolation)

- Programmatic account/mailbox provisioning (Microsoft Graph, Google Workspace Admin SDK, SCIM).
- JMAP session model and `primaryAccounts` (RFC 8620/8621).
- String slugification for identifier synthesis.
- Idempotent "create-or-409" API patterns.
- Externalized secrets and encrypted credential storage (Vault, Key Vault, K8s Secrets); Twelve-Factor Config/Disposability.

Candidate-novel (the disclosed combination)

- **Two-id resolution** distinguishing provisioning- vs. session-account-identifier for *machine* identities, stored as distinct fields.
- **Credential rehydration on boot** into an in-process client cache, with the JMAP username **reconstructed from the address local-part**, restoring a whole agent fleet via a single store read and **without** any provisioning-API call.
- The **unified machine-paced lifecycle** binding lazy domain creation, collision-safe synthesis, two-id resolution, and rehydration with explicit fail-open/closed posture.

Honesty attestation

The prior-art searches behind this document are **directional, not exhaustive**. They reflect a good-faith review of public standards, vendor documentation, and common engineering patterns as of the publication

date. No claim is made that every relevant reference has been found. The purpose is defensive disclosure: to establish that these techniques were publicly described as of 2026-06-25.

Appendix B — Glossary

Term	Definition
Persona / Agent	An autonomous AI actor instantiated by the platform, treated as a first-class identity.
Slug	A normalized, lowercase, separator-collapsed token derived from a human-readable name.
Local-part	The portion of an email address before the `@`.
JMAP	JSON Meta Application Protocol (RFC 8620), and JMAP for Mail (RFC 8621).
Provisioning-account-identifier (`pid`)	The id returned by the admin account-creation API; used for admin operations.
Session-account-identifier (`sid`)	The id under `primaryAccounts` in a JMAP session; required for JMAP method calls.
App-password	A per-account secret used to authenticate JMAP operations.
Rehydration	Reloading persisted credentials from a durable store into the in-process cache at startup.
Lazy domain creation	Creating the tenant mail domain just-in-time on the first agent that needs it.
Fail-open / fail-closed	Whether a failure permits continued operation (open) or halts it (closed).

Appendix C — Reference-Implementation Index

Path	Role
`src/README.md`	What the sample shows, how to run it, clean-room disclaimer.
`src/mailbox-provisioner.js`	The Mailbox Manager: slugify, ensureDomain, provision, resolveSessionAccountId, deprovision, rehydrateCredentials.
`src/fake-jmap-server.js`	In-memory JMAP/admin stand-in with a session id distinct from the provisioning id.
`src/demo.js`	Runnable end-to-end scenario with a restart-and-rehydrate self-check.

Appendix D — Defensive-Publication Deposit & Timestamp

This document is published as a **Technical Defensive Publication** by **Gus IT LLC (Florida, USA)**, authored by **Gustavo Assuncao, PhD**, with a publication date of **2026-06-25**, version **1.0**.

Deposit: to be assigned (IP.com / Zenodo / arXiv) — establishing a public, dated, citable prior-art record. No DOI is asserted at the time of writing.

Intent: this disclosure is **intentionally public** so as to constitute prior art that **bars later patenting of the disclosed techniques by others**, and to keep those techniques freely practiceable by everyone, including the publisher. Licensed under AGPL-3.0-or-later (see LICENSE and NOTICE), whose express patent grant reinforces this defensive intent.